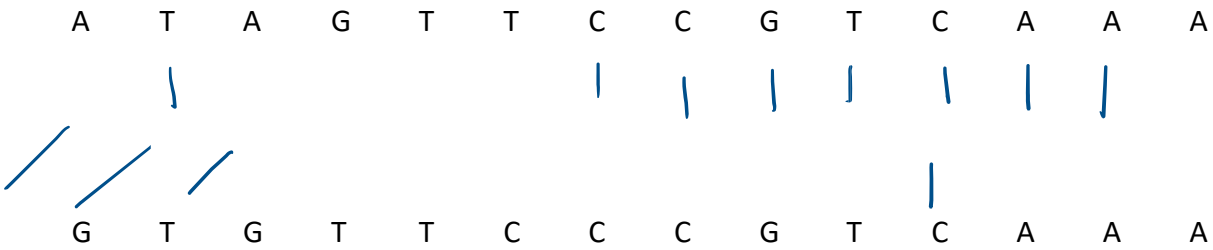
Assignment: Dynamic Programming

1. **Solve a problem using top-down and bottom-up approaches of Dynamic Programming technique**

   DNA sequence is made of characters A, C, G and T, which represent nucleotides. A sample DNA string can be given as 'ACCGTTTAAAG'. Finding similarities between two DNA sequences is a critical computation problem that is solved in bioinformatics.

   Given two DNA strings find the length of the longest common string alignment between them (it need not be continuous). Assume empty string does not match with anything.

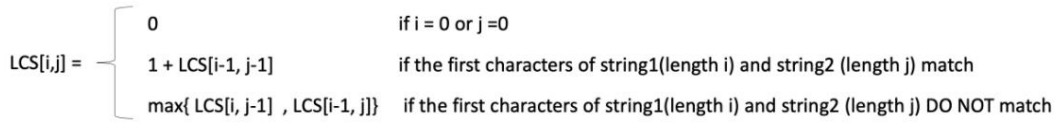
   Example: DNA string1: ATAGTTCCGTCAAA ; DNA string2: GTGTTCCCGTCAAA



   Length the best continuous length of the DNA string alignment: 12 (TGTTCCGTCAAA)

   a. Implement a solution to this problem using Top-down Approach of Dynamic Programming, name your function **dna_match_topdown(DNA1, DNA2)**
      Answer: in python file
   b. Implement a solution to this problem using Bottom-up Approach of Dynamic Programming, name your function **dna_match_bottomup(DNA1, DNA2)**
      Answer: in python file
   c. Explain how your top-down approach different from the bottom-up approach?
      Answer: Both approaches are using a two-dimensional matrix to the store the match counts of the LCS. The top-down approach uses recursion with memorization to save the value of the completed sub-tasks, which will be used to check if a sub-task was complete and not to run it again. The bottom-up approach uses iteration and fills in each of the indexes in the two-dimensional matrix once. It does this by comparing if the index of dna1 matches dn2 and adds one from the diagonal top left slot of the current slot in the two-dimensional matrix.
   d. What is the time complexity and Space complexity using Top-down Approach
      Answer: O(m*n)
   e. What is the time complexity and Space complexity using Bottom-up Approach
      Answer: O(m*n)
   f. Write the subproblem and recurrence formula for your approach. If the top down and bottom-up approaches have the subproblem recurrence formula you may write it only once, if not write for each one separately.
      Answer:

The subproblem and recurrence relation formula is the same for both and is the same as the LCS version. DNA1 of size n and DNA2 of size m will have subproblems of DNA1 of size n-1 and DNA2 of size m-1



Name your file **DNAMatch.py**

2. Solve Dynamic Programming Problem and Compare with Naïve approach

You are playing a puzzle. A random number N is given, you have blocks of length 1 unit and 2 units. You need to arrange the blocks back to back such that you get a total length of N units. In how many distinct ways can you arrange the blocks for given N.

a. Write a description/pseudocode of approach to solve it using Dynamic Programming paradigm (either top-down or bottom-up approach)

```
def puzzle_bottom_up(n):

    # these are the first base cases that need to be established
that can exit the code if these are what are passed
    if n < 1:
        return 0
    if n == 1:
        return 1
    if n == 2:
        return 2

    # create a puzzle_table the size of n and assign index 0 as 1
and index 1 as 2 because these are the base cases for
    # the first two inputs of n
    puzzle_table = [0] * (n)
    puzzle_table[0] = 1
    puzzle_table[1] = 2

    # the current index puzzle_table[i] is the sum of the previous
two indexes
    for i in range(2, n):
        puzzle_table[i] = puzzle_table[i-1] + puzzle_table[i-2]

    # return puzzle_table
    return puzzle_table[i]
```

b. Write pseudocode/description for the brute force approach

```
def puzzle(n):

    # this is the brute force approach in which we repeat each
subproblem multiple times even when they occurred by the
    # iterative approach without memoization
    if n < 1:
        return 0
    if n==1:
        return 1
    if n==2:
        return 2
```

```
    return puzzle(n-1) + puzzle(n-2)
```

c. Compare the time complexity of both the approaches

Brute Force: O(2^n) because subproblems are repeated

Bottom-up: O(n) This is because we iterative through an array 1 time and don't repeat subproblems

d. Write the recurrence formula for the problem

Similar to the Fibonacci sequence

$$F(n) = \begin{cases} 1 & \text{If n = 1} \\ 2 & \text{If n = 2} \\ F(n-1) + F(n-2) & \text{otherwise} \end{cases}$$

Example 1:

Input: N=2, Result: 2

Explanation: There are two ways. (1+1 , 2)

Example 2:

Input: N=3, Result: 3

Explanation: There are three ways (1+1+1, 1+2, 2+1)

Debriefing (required!): ----------

Report:

Fill the report in the Qualtrics survey, you can access the link here.
(https://oregonstate.qualtrics.com/jfe/form/SV_6As0QKSDULDVnLg)

Note: 'Debriefing' section is intended to help us calibrate the assignments.